计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210100110

半虚拟化框架 Virtio 下的实时网络 I/O 请求门控机制

申浩希　牛保宁
太原理工大学信息与计算机学院　太原 030024
(444969304@qq.com)

摘　要　响应时间是服务等级目标(Service Level Objective,SLO)的一个重要性能指标,与资源的使用量有关。资源充足可以保证请求的正常执行,响应时间短;资源不足,请求需要等待资源,响应时间长。在云计算虚拟化环境下,控制资源的访问既有对整体资源的控制,也有对 CPU、网络带宽等单个资源的控制,但是目前很少有通过对网络 I/O 请求的直接控制来保证响应时间。为了获得更好的性能,虚拟化技术大多采用半虚拟化框架 Virtio。网络 I/O 请求通过 Virtio 共享通道进行传输,使得在 Virtio 设立网络 I/O 请求的门控机制成为可能。文中利用双端聚合方法(Two-end Aggregation Method,TAM),提出实时网络 I/O 请求门控机制(Gating Mechanism for Real-time Network I/O Requests,GMRNR),通过控制网络 I/O 请求经过 Virtio 的时刻,保证各类请求的响应时间。GMRNR 设立在 Virtio 前端 virtio-net 模块中,将请求按照其响应时间指标分级,采用计时器和聚合队列长度来控制不同级别请求经过 Virtio 的时刻和聚合频率,保证请求的响应时间。实验测试表明:GMRNR 能够区分网络 I/O 请求优先级,在资源充足时,使得不同等级的网络 I/O 请求在各自要求的时间内完成;在资源不充足时,能优先保证高优先级的网络 I/O 请求的响应时间。同时,GMRNR 具有较高的资源利用效率。

关键词:响应时间;服务等级目标;网络 I/O 请求;门控机制;Virtio;优先级
中图法分类号　TP391

Gating Mechanism for Real-time Network I/O Requests Based on Para-virtualization Virtio Framework

SHEN Hao-xi and NIU Bao-ning
School of Information and Computer Science,Taiyuan University of Technology,Taiyuan 030024,China

Abstract　Response time is an important performance indicator of the service level objective (SLO),which is related to the usage of resources. If resources are sufficient to ensure the normal execution of the request,the response time is short. If resources are insufficient,the request needs to wait for resources,and the response time is long. In the cloud computing virtualization environment,the control of resource access includes both the control of the overall resource and the control of individual resources such as CPU and network bandwidth. However,there are currently few direct control of network I/O requests to ensure response time. In order to achieve better performance,virtualization technology mostly uses the para-virtualization framework Virtio. Network I/O requests are transmitted through the Virtio shared channel,making it possible to set up a gating mechanism for network I/O requests in Virtio. Therefore,the study uses the two-end aggregation method (TAM) to propose gating mechanism for real-time network I/O requests (GMRNR),which controls the time when the network I/O request passes Virtio to ensure the response time of various requests. GMRNR is set up in the virtio-net module of Virtio front-end and classifies requests according to their response time indicators. It uses timers and aggregation queue length to control the time and aggregation frequency of different levels of requests through Virtio to ensure the response time of the request. Experimental tests show that GMRNR can distinguish the priority of network I/O requests,and when resources are sufficient,network I/O requests of different levels can be completed within their respective required time. When resources are insufficient,the response time of high-priority network I/O requests is given priority. Meanwhile,GMRNR has high resource utilization efficiency.

Keywords　Response time,Service level objective(SLO),Network I/O requests,Gating mechanism,Virtio,Priority

到稿日期:2021-01-14　返修日期:2021-05-31　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(62072326);山西省重点研发计划项目(201903D421007)
This work was supported by the National Natural Science Foundation of China(62072326) and National Key Research and Development Plan of Shanxi Provence(201903D421007).
通信作者:牛保宁(niubaoning@tyut.edu.cn)

1 引言

云计算是一种计算服务[1-3]，相比传统 IT 服务模式，可以保证用户需求的即时性，基于即用支付收费模式（pay-as-you-use），通过动态灵活的资源池形式，以标准或定制的计算服务租用模式，为用户提供各种可虚拟扩展的在线业务形式[4-5]。其中，服务等级目标（SLO）[6]规定了云计算服务提供商提供服务时的服务质量（Quality of Service，QoS）[7]。响应时间（Response Time，RT）是 SLO 的一个重要指标[8-9]。

请求的响应时间和资源的使用量有关[10]。分配的资源充足，可以保证请求的执行，响应时间短；分配的资源不足，请求需要等待资源，响应时间长。因此，如何分配云计算服务的资源成为保证请求的响应时间、满足 SLO 的一项关键技术。在已有研究中，保证响应时间的方法有两类：第一类是控制资源使用量来保证响应时间[11-12]。在资源充足时，该方法可以保证请求的响应时间，而在资源不足时，无法保证请求的响应时间。这类方法通常控制系统总的资源量，很难区别对待不同要求的请求。另一类是控制请求对资源的访问来保证响应时间[13]，其能够区别对待不同类别的请求。这类方法可以直接影响请求响应时间，通常被称为门控机制[14]。其可以是针对整体资源访问的控制，如虚拟机访问控制机制[15]；也可以是针对单个资源的控制，例如控制请求对 CPU[16]、网络带宽[17]等的使用。

本文探讨云计算环境下，采用控制请求对网络 I/O 资源的访问，保证请求响应时间的方法。在虚拟机环境下，请求对 I/O 的访问需要经过虚拟机监视器 VMM[18]，这提供了对 I/O 请求进行门控的机会。因此，本文基于半虚拟化框架 Virtio[19-20]，研究虚拟机网络请求控制机制，利用双端聚合方法（Two-end Aggregation Method，TAM）[21]提出一种保证不同级别请求响应时间的机制——实时网络 I/O 请求门控机制 GMRNR。在 virtio-net 模块中设立 GMRNR，当网络 I/O 请求经过 Virtio 前端的 virtio-net 模块时，GMRNR 采用计时器[22]和聚合队列长度控制不同级别的网络 I/O 请求通过 Virtio 的时刻和聚合频率，从而保证请求响应时间。计时器用来设定网络 I/O 请求聚合传输的时刻，该时刻的设定依据请求的响应时间要求。同时，聚合队列也根据其长度控制队列聚合的频率，请求的优先级越高，聚合队列长度越短，聚合频率就越高。当网络 I/O 请求进入门控机制后，通过分发器依据响应时间要求进行优先级分级[23]，优先级较高的请求在密集性高时聚合传输，在密集性低时则单独执行，以保证其实时性。优先级较低的请求放入聚合队列等待，待其中任一请求即将达到要求的响应时间或等待的请求达到聚合队列长度则进行聚合传输，保证其响应时间。通过分类聚合的方式，共享网络请求的资源，减少系统切换并降低 CPU 开销，以保证各级别的请求响应时间。

2 相关工作

本节首先讨论云环境下保证请求响应时间的方法和优先级分类思想，然后介绍 Virtio 框架。

2.1 保证请求响应时间的方法和优先级分类思想

请求的响应时间与资源的使用量有关。保证请求响应时间的方法分为两类：控制请求对资源的访问和控制分配给请求的资源量。

（1）控制请求对资源的访问

在控制请求对资源的访问方面，可以通过对整体资源的控制，来限制请求对资源的访问，也可以控制请求对单个资源的访问。其中，在整体资源的访问控制方面，Popa 等[15]提出一种基于 hypervisor 的多租户访问控制机制 CloudPolice，帮助 hypervisor 动态地协调它所承载的虚拟机的访问控制策略。Qiu 等[24]提出一种基于多级机器学习的资源管理框架，该框架使用跟踪协调器，为每个微服务实例收集跟踪数据，并存储在数据库中，检测 SLO 违规问题，根据资源预测算法控制请求延迟，降低违约率。而控制请求对单个资源的访问中，Zhang[16]提出基于操作系统的最早截止时间优先调度算法（Load-balance Earliest Deadline First，LEDF），该算法在简单优先调度算法的基础上增加了负载均衡策略，分配的虚拟机 CPU 资源随着请求绝对截止期限的变化而改变，可以在混合应用环境中控制最大响应时间，保证请求的响应时间。Zhou[17]提出虚拟机网络带宽动态调节机制，由用户层的带宽动态分配策略依据响应时间要求在驱动层分配带宽。Lee 等[25]提出一种调度器 ISACS，它基于 Credit2 调度程序，目的在于控制 I/O 操作次数，当 I/O 操作次数过多时会产生 vCPU 帮助执行任务，通过控制 I/O 操作次数来增强性能，保证请求响应时间。本文的机制就是对单个资源访问的控制，但是不需要控制 CPU 等资源，而是利用 Virtio 框架在其内部设立门控机制，直接对网络请求进行控制，相比对 CPU 等资源访问的控制更为直接、方便。

（2）控制分配给请求的资源量

在控制资源使用量方面，可以依据请求的多少即需求量分配资源，即请求越多，分配到的资源也就越多。Sun 等[11]提出最小访问代价的数据复制策略（MACR），通过计算共享资源的全局访问代价，从全局角度以少量数据副本获得平均访问延迟最小目标下的副本优化分布，从而保证请求的响应时间。Hou 等[12]提出一种基于资源使用阈值边界的虚拟机分配方法，建立了资源使用效率阈值边界，通过阈值判断服务器的负载情况，从而迁移服务器资源来控制请求响应时间。Zhang 等[26]提出一个通用的推理服务系统 MArk 模型，以实现服务等级目标的承诺和花费效益。为了提高性能成本比，MArk 选择带有预测性的自动扩展以低成本的方式隐藏供应延迟。Zhou 等[27]提出一种细粒度资源调度方法，该方法根据相似任务运行的信息推测任务资源需求，将任务划分为若干份来执行，从分配资源和时间两方面细化资源分配粒度，通过控制分配给请求的资源大小来影响请求响应时间。本文机制则不考虑资源的分配问题，而是在资源内部对请求进行控制。

优先级分类可以帮助区分不同用户请求的响应时间

目标，请求的优先级分类思想也有助于门控机制的建立。Su等[23]提出多特征优先级实时调度任务模型MDPSS，任务优先级动态变化。当高优先级请求到达后，低优先级请求会随着请求等待时间的增加而提升优先级，从而率先利用资源执行任务。Liu[28]为虚拟CPU添加高优先级调度状态——boost状态，当有高优先级网络I/O请求到达后直接进入boost状态，请求就会被最先处理，而低优先级请求会根据请求的响应时间要求目标随着等待时间增加而变为高优先级，随后进入boost状态进行处理，充分保证了高优先级请求的响应时间。Rajavel等[29]利用CPU调度器将先到先服务策略和循环调度相结合，检测队列中即将到达要求时间的请求，将其提升至高优先调度层级率先调度执行。

网络性能优化也有助于控制请求的响应时间。在网络性能优先的研究中，Liu等[21]提出双端聚合方法(Two-end Aggregation Method，TAM)，对Virtio前后端网络请求传输优化，请求经过前端驱动逐一处理后置于共享通道，进行系统切换，后端统一接收后逐一处理，通过将Virtio前、后两端的数据进行聚合传输来减少超级调用次数，从而缩短响应时间。

根据以上研究，本文借助TAM方法在Virtio前段virtio-net模块中建立实时网络I/O请求门控机制(GMRNR)，控制请求访问底层物理资源，达到保证各级别请求响应时间的目的。

2.2 Virtio框架

虚拟化性能主要体现在I/O任务的处理，KVM虚拟化的I/O服务都由QEMU处理[30]。在网络虚拟化环境中，原生KVM虚拟机在磁盘、网络等I/O设备的访问性能非常低，仅满足于实验测试场景的应用。为了使虚拟机获得较好的I/O性能，必须抛弃QEMU提供的完全虚拟化的模拟设备驱动。因此，目前I/O性能优化的解决方案是Virtio设备驱动标准下的半虚拟化I/O驱动[29-32]。KVM半虚拟化框架如图1所示。

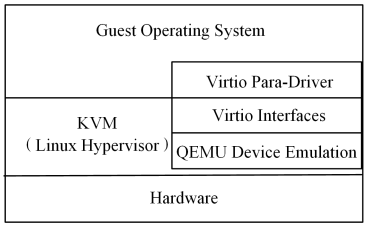

图1 KVM半虚拟化框架

Fig. 1 KVM para-virtualization framework

Virtio是一种通用的驱动分离设备模型，包括前端驱动程序、后端驱动程序以及前后端信息传输的共享通道。该模型为网络设备的处理提供了解决方案。TAM方法在前端virtio-net模块中将网络I/O请求放入聚合队列等待，待请求聚合的大小达到1024字节时，则进行聚合传输，执行一次超级调用。而后端则是取出队列请求依次处理，在处理完队列中全部请求后再执行一次超级调用，通知前端，如图2所示。

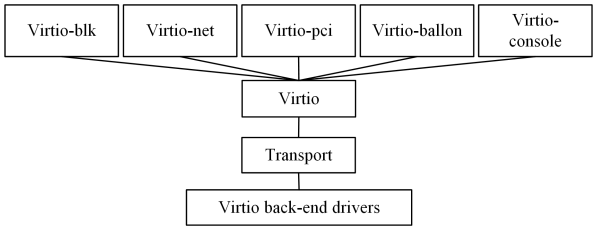

图2 Virtio半虚拟化框架

Fig. 2 Virtio para-virtualization framework

本文门控机制建立在Virtio前端的virtio-net模块中，借助TAM方法的前端聚合方法聚合网络I/O请求，控制不同级别网络I/O请求通过Virtio的时刻，保证各级别请求的响应时间。

3 实时网络I/O请求门控机制(GMRNR)

3.1 问题定义和GMRNR的思路

问题定义：网络I/O请求优先级与请求响应时间要求有关，按照优先级进行分级：

(1)较高优先级请求应该率先保证其响应时间，保证请求的实时性；

(2)较低优先级请求尽量保证其响应时间即可，以提高资源利用率。

本文引入TAM方法，该方法的聚合思路有助于本文的门控机制更有效地执行，无论网络I/O请求密集与否，GMRNR均可以保证较高优先级请求的响应时间。

面对响应时间要求不同的用户，优先级的判定极为关键。以用户要求为基准，本文定义用户请求优先级为P_q，用户支付费用为M，T_r表示用户要求请求平均响应时间。则优先级可表示为：

$$P_q=\frac{M}{T_r} \tag{1}$$

其中，支付费用与要求请求平均响应时间的比值越大，则优先级越高。通过设定不同临界值，判定请求的优先级。

例如，设定两类临界值，将请求优先级根据临界值P_1和P_2划分为高优先级S级、中优先级A级、低优先级B级3类(P_1代表高、中优先级的分界值，P_2代表中、低优先级分界值)，则：

(1)当$P_q \geqslant P_1$时，将此类请求设为高优先级请求S级请求；

(2)当$P_1 > P_q \geqslant P_2$时，将此类请求定义为中优先级请求A级请求；

(3)当$P_q < P_2$时，将此类请求定义为低优先级请求B级请求。

GMRNR的思路是：在虚拟化中，面对不同的请求，借助TAM聚合方法在执行网络I/O密集型任务时减少客户机和宿主机的上下文切换，将多个请求聚合传输，使得多次超级调用合并为一次，从而降低能耗，率先保证较高优先级的请求响应时间，较低优先级请求则聚合等待，尽量

保证较低优先级的请求响应时间。

首先，针对 Virtio 前、后端的聚合进行分析。本文机制主要在 Virtio 前端的 virtio-net 模块中实现，将机制设立于前端更有助于对网络请求的直接控制，同时，由于本文提出的门控机制针对的是不同优先级用户，因此本文借助 TAM 方法的前端聚合法对网络请求进行聚合，暂不考虑后端聚合。

其次，由于需要保证较高优先级请求的实时性，因此，对较高优先级请求是否聚合进行分析。一般以网络 I/O 请求的密集程度判断请求传输方式。首先，定义超级调用频率 f_c，即在单位时间内实际执行的超级调用次数。通过动态调整 f_c 来控制请求的处理延迟。然后定义单位时间内网络 I/O 请求的频率 f_i。从总体上来说，当 f_i 较高时，通过降低 f_c，也就是聚合请求减少超级调用次数，可以提高请求响应时间，减少 CPU 能耗；当 f_i 较低时，通过提升 f_c，以保障用户请求的要求响应时间。假设一个请求进入队列缓存的时间固定为 T_q，可以将一次超级调用的消耗时间固定为常数 T_c，那么在队列缓存状态下，第 n 个 I/O 请求到达时，全部聚合等待的时间总和为：

$$T_n=(1+2+\cdots+n)\left(T_q+\frac{1}{f_i}\right) \tag{2}$$

如果单位时间内 I/O 请求的频率为 f_i，则可以判断下一个请求到达的时间为 $\frac{1}{f_i}$。

根据式(2)，需要确定 Virtio 前端驱动接收网络 I/O 请求时是否应等待下一个请求聚合或者执行超级调用通知后端发送数据。

设当第 n 个请求到达队列时，n 个请求的等待时间为 T_n，立即进行聚合传输，执行一次超级调用，那么请求平均响应时间为：

$$T_n'=\frac{T_n+T_c+nT_q}{n} \tag{3}$$

这时需要判断是否执行超级调用，如果选择等待下一个 I/O 请求到达再进行聚合，经过 $\frac{1}{f_i}$ 的时间，在第 $n+1$ 个 I/O 请求到达时，这 $n+1$ 个请求的平均响应时间为：

$$T'_{n+1}=\frac{T_n+T_c+(n+1)T_q+(n+1)\frac{1}{f_i}}{n+1} \tag{4}$$

如果在收到第 n 个 I/O 请求时直接执行超级调用，则第 $n+1$ 个的请求的响应时间至少为 T_c+T_q，那么这 $n+1$ 个请求的平均响应时间则为：

$$T''_{n+1}=\frac{T_n+2T_c+(n+1)T_q}{n+1} \tag{5}$$

由上式可得，当 $T'_{n+1}>T''_{n+1}$ 时，即 $(n+1)\frac{1}{f_i}>T_c$ 时，不执行超级调用继续等待下一个 I/O 请求会增加响应时间，因此应立即执行超级调用；当 $T'_{n+1}<T''_{n+1}$ 时，即 $(n+1)\frac{1}{f_i}<T_c$ 时，立即执行超级调用会增加响应时间，则应等待下一个 I/O 请求到来后再执行超级调用。

其中，T_q 和 T_c 的时间根据实际测试得出，$\frac{1}{f_i}$ 则需要进行比较，如果 $\frac{1}{f_i}$ 大于计时器设定的时刻最大值，则进行超级调用。若下一个请求到达的 $\frac{1}{f_i}$ 超出设定的时刻，则立即执行超级调用；反之，则判断放入队列缓存请求时是否 $T'_{n+1}>T''_{n+1}$，如果在等待过程中超过设定的时刻，则直接执行超级调用，否则等待 $T'_{n+1}>T''_{n+1}$ 再执行。

本文方法对较高优先级请求是否聚合进行了判断，最大限度地对网络请求进行聚合，在资源饱和时降低资源的消耗。

3.2 优先级分类

为了便于讨论以及后续实验，本文将网络 I/O 请求按照问题定义中的 3 类优先级进行划分。

(1)S 级请求即高优先级请求，其针对重要用户将请求定义为最优处理，即在任何情况下率先保证其请求的响应时间。若资源充分，则最快速地保证其响应时间；若资源不足，则按照其内部先后到达顺序，以先到先服务的原则进行处理。原则上在执行 S 级请求期间，不考虑其他级别请求的执行。S 级请求是否聚合根据请求到达的频率来判定，如果频率较高，则进行聚合，以减少资源的浪费；若请求到达频率较低，则不进行聚合，直接传输，以加快响应。

(2)A 级请求即中优先级请求，主要针对会员用户，要求是保证请求的响应时间即可。即在没有执行 S 级请求时优先执行 A 级请求。在 S 级请求连续执行时根据 A 级请求聚合的情况判断是否中断 S 级传输过程，保证 A 级请求的响应时间。在 A 级请求到达时，不采用直接传输的方式，而是将其放入等待队列，根据 S 级请求的执行情况以及 A 级请求要求的响应时间进行聚合执行。

(3)B 级请求即低优先级请求，主要受众为免费用户，其请求内容一般为不关键请求，只需尽力保证其响应时间即可。在没有 S 级、A 级请求执行的情况下，执行 B 级请求。若 S 级、A 级请求连续执行，则不考虑 B 级请求的执行。若 S 级、A 级请求不连续执行，B 级请求则聚合等待执行，在资源空闲时进行传输。

3.3 GMRNR 的内部架构和处理流程

GMRNR 内部架构如图 3 所示，其核心内容为排队器的设立。

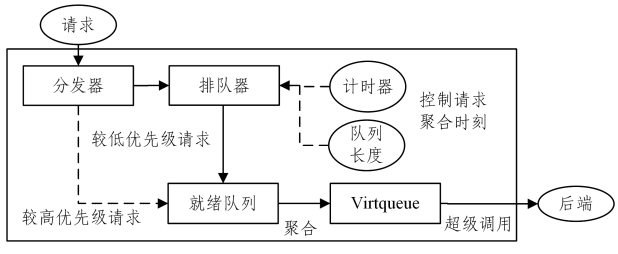

图 3 GMRNR 内部架构

Fig. 3 GMRNR internal framework

当网络 I/O 请求发出，首先进入分发器进行判断，根据请求的来源即虚拟机 IP 地址判断同一优先级级别请求，按照

优先级分类，S级直接进入就绪队列等待，也就是进入缓存队列中判断是否等待聚合或者直接执行超级调用，随后接收中断，完成响应。剩余较低优先级A级、B级请求则进入排队器分别等待，根据各自聚合队列的聚合程度或者计时器的时刻以及S级请求是否正在发送，从而判断能否将等待的A级、B级队列请求聚合，执行超级调用。

低级请求聚合通过判断上一级网络请求是否正在发送、计时器是否到达指定时刻或聚合队列是否达到聚合程度来判定是否执行超级调用。当其中一种条件满足，即可判断队列为可发送状态，再去执行超级调用并通知后端。GMRNR I/O请求处理流程如图4所示，其主要步骤如下。

(1)S级I/O请求到达，将I/O请求放入缓存队列，增加队列大小，并计算与前一个请求I/O相差的时间，从而计算出请求的频率f_i。

1)判断$\frac{1}{f_i}$是否大于限时阈值，如果大于，则执行virtqueue_kick函数并完成调用，否则不执行。

2)判断是否$T'_{n+1}>T''_{n+1}$，如果大于或者达到限时计时器限制时间，则执行超级调用，执行virtqueue_kick函数。

(2)A级I/O请求到达时：

1)判断到达请求是否达到队列长度，当$i>=$sq_size时则执行A级请求中断，使B级请求先执行virtqueue_kick函数。

2)判断S级设备队列状态，在S级请求频率f_i达到计时器设定的频率且未发出新的请求时，则A级队列执行virtqueue_kick函数。

3)判断计时器时刻，在A级队列计时器达到其中一个请求的最长响应限制时间的时刻，执行超级调用，优先执行virtqueue_kick函数。

(3)B级I/O请求到达时：

1)判断到达请求是否达到队列长度，当$i>=$sq_size时则执行高、中优先级中断，使低优先级先执行virtqueue_kick函数。

2)判断S级、A级设备队列状态，在S级请求频率f_i达到计时器设定的频率且未发出新的请求，或A级队列未达到发送状态时，聚合B级队列执行virtqueue_kick函数。

3)判断计时器状态，在B级队列计时器达到其中一个请求的最长响应限制时间的时刻，执行超级调用，优先执行virtqueue_kick函数。

(4)超级调用通知后端，结束并返回。

通过上述步骤，完成函数的实现。通过限制计时器的时间和队列大小，并且判断频率值，来执行超级调用，处理I/O请求。

基于此，设立实时网络I/O请求门控机制GMRNR，通过计时器和聚合队列长度控制请求的聚合时刻和聚合频率，保证较高优先级请求的响应时间，提高较低优先级请求的资源利用率，降低能耗。

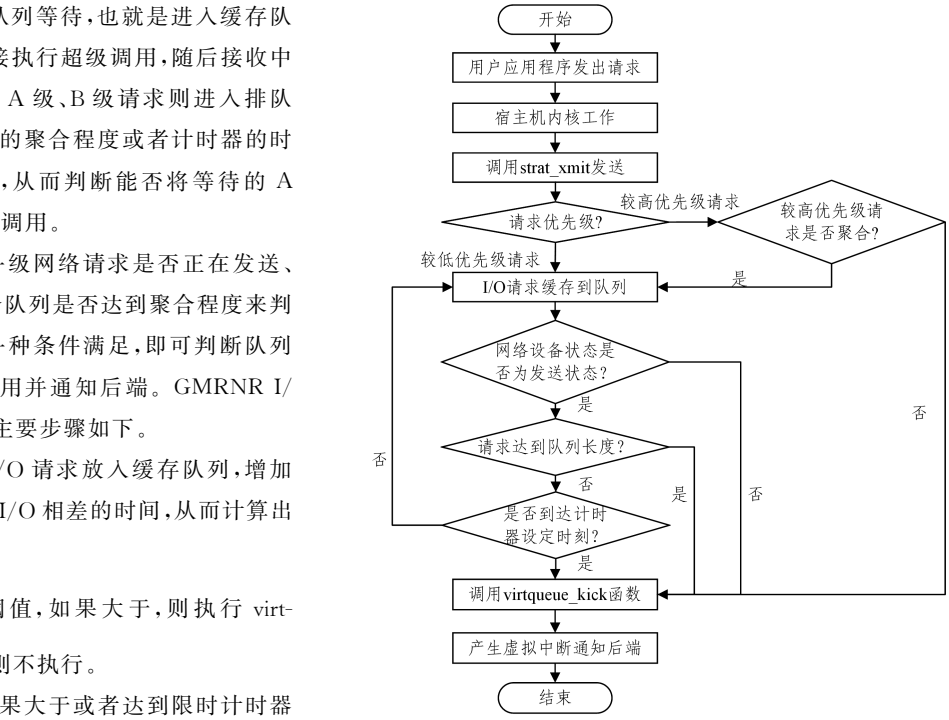

图4 GMRNR I/O请求处理流程

Fig. 4 GMRNR I/O requests workflow

4 实验分析

4.1 实验环境与设计

(1)实验环境

硬件环境如下：处理器为Intel(R)Xeon(R) Bronze 3104 CPU @17.0GHz (2处理器)；内存为64GB。

软件环境如下：宿主机操作系统为Ubuntu 18.04.4 LTS；宿主机内核版本为Linux 5.4.0-52-generic；Qemu-kvm版本为3.1.0；客户机系统为Ubuntu 18.04.4 LTS；客户机内核版本为Linux 5.4.0-42-generic x86_64。

(2)实验设计

为了验证GMRNR门控机制有一定的适用性，需要分别验证优先级分类和请求门控机制对性能的提升，因此实验选择LEDF[14]，ISACS[23]，BOOST[26]以及MDPSS[21] 4种方法作为对比方法。

LEDF和ISACS方法在不区分优先级的情况下进行控制，主要以到达顺序为目标进行控制。

BOOST和MDPSS方法则是不对请求进行直接控制，仅按照请求的优先级执行响应。

实验1 GMRNR有效性测试

实验中，本文方法与LEDF和ISACS方法以及BOOST和MDPSS方法进行了对比，以验证门控机制进行优先级分类的有效性以及直接控制请求的必要性。以规定的Class S为S级优先级请求保证的响应时间目标，Class A为A级优先级请求保证的响应时间目标。在规定目标的基础上，利用Ping测试每个虚拟机向宿主机发送1 000次http请求的响应时间。

实验 2　性能开销测试

为了验证 GMRNR 在保证各类请求响应时间时并没有消耗更多的资源，同时它的控制机制通过控制上下文切换次数和系统中断次数减少了 CPU 开销，达到了降低服务提供商成本的目的，实验用 vmstat 检测工具对 Netserver 服务端的宿主机进行整体性能监控，监测 CPU 的各项开销。

实验 3　网络访问性能测试

为了验证 GMRNR 通过网络传输访问本地磁盘的性能，仍然以 Class S 和 Class A 为响应时间目标，对 5 种方法进行对比实验。通过虚拟机同时向宿主机发出读、写本地磁盘文件的网络请求并返回，文件大小分别为 128 kB，256 kB，512 kB，1 MB，2 MB，4 MB 以及 8 MB。实验以访问平均 1MB 的文件大小的响应时间为例。

4.2　实验结果与分析

(1)GMRNR 有效性测试

从图 5 可以看出，随着虚拟机数量的增加，LEDF 由于负载均衡的作用，控制最早截止时间优先调度的请求，根据 LEDF 按照请求到达的先后顺序进行控制，但是并没有依据优先级顺序响应请求，因此，它的 3 类优先级请求的响应时间差别不大，在虚拟机数量相对较少时可以保证请求的响应时间达到 Class A 级别的要求，但是后来由于 CPU 饱和并且任务请求密集，响应时间增加，当饱和状态持续时，其响应时间也趋于平稳。

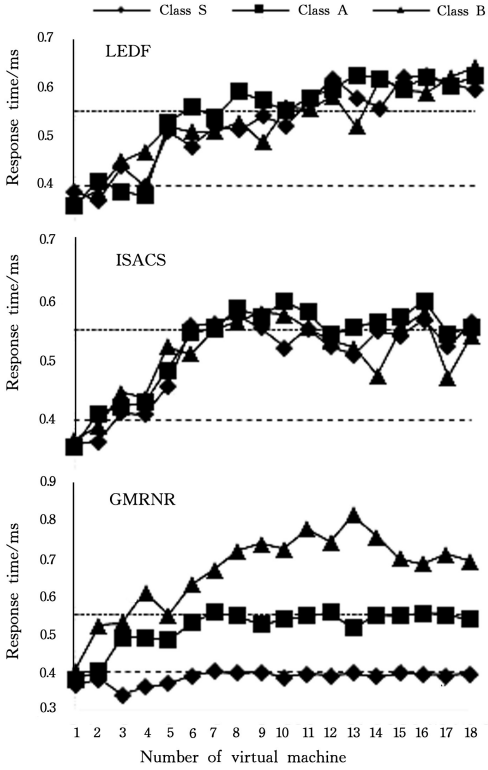

图 5　LEDF，ISACS 和 GMRNR 方法下的 3 类优先级请求响应时间

Fig. 5　Response time of three priority requests for LEDF，ISACS and GMRNR

ISACS 由于并不执行优先级顺序，因此在开始阶段和 LEDF 相似，响应时间随着资源不足而不断增加，但是到达一定的程度后它的控制机制会创建出新的虚拟 CPU 供其使用，减少了请求集中带来的过多开销，因此请求响应时间有所下降，并且部分请求已经达到了 Class A 级别，但是由于其不区分优先级，因此 3 种类别的请求的响应时间相差不大。GMRNR 相比 LEDF 和 ISACS 具有优先级顺序机制，在保证 S 级请求响应时间的情况下，结合机制的聚合作用，同时保证了 A 级优先级请求的响应时间。而 B 级请求的响应时间相比以上两种方法也有所降低。

从图 6 可以看出，BOOST 由于按照请求优先级的顺序执行，在资源充足时，S 级请求能够得到充分保证，A 级请求的响应时间也基本上可以得到保证。但在 CPU 趋于饱和后，由于资源受到限制，请求延时增加。而 MDPSS 也用到了优先级，并且根据优先级顺序依次处理请求，因此与 BOOST 起初较为相似，但是后期由于它具有优先级机制，等待时间长的请求优先级增高，部分低优先级请求升为高优先级请求被优先处理，使得高优先级部分请求的响应时间降低，但是中优先级请求的响应时间因优先处理高优先级请求而不断增加。

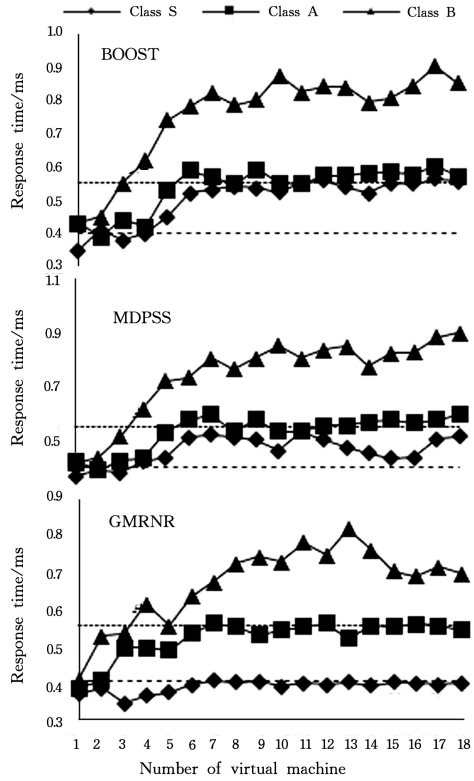

图 6　BOOST，MDPSS 和 GMRNR 方法下 3 类优先级请求的响应时间

Fig. 6　Response time of three priority requests for BOOST，MDPSS and GMRNR

图 6 中，BOOST 和 MDPSS 两种方法由于控制机制缺少合理的控制，使得请求在 CPU 达到饱和后响应时间延长，各类请求的响应时间可能均难以保证。而 GMRNR 在区分优先级级别的情况下更为直接地控制性能。在保证 S 级请求响应时间的情况下，其控制机制由于聚合请求减少了上下文

切换和超级调用次数，使得 A 级请求的响应时间得到保证。而 B 级请求也因为控制机制的聚合能力，在资源饱和的状态下仍然可以做到尽量响应。相比 BOOST 和 MDPSS，GMRNR 中 B 级请求的响应时间分别降低了 18.6 和 23.5%。

通过对比可以看出，由于 GMRNR 的控制机制在区分优先级的同时对请求进行聚合，因此在资源饱和的状态下仍然可以保证较好的性能，从而验证了 GMRNR 进行优先级分类聚合的有效性。

(2)性能开销测试

对 Netserver 服务端的宿主机系统进行整体性能测试监控，监测 CPU 各项开销。监测所用工具为 vmstat。

由表 1 可知，相比其他 4 种方法，GMRNR 降低了 CPU 等待 I/O 资源消耗的百分比，充分利用了资源。由于控制机制，用户进程占用 us 的百分比增加了，使用率有所提高。此外，由于控制机制的聚合作用减少了 CPU 多余的开销，系统 CPU 使用率有所降低，发生中断的次数 in 也有所减少，并且系统上下文切换次数也有所减少。但是上下文切换次数仍然保持在很高的一个状态，并且与 MDPSS 接近，原因主要是在批量网络数据传输的情况下，为了保证高优先级请求的传输，高优先级聚合队列的长度和计时器时长设定得较短。而 MDPSS 由于本身动态调节优先级的性能，增加了低优先级请求处理的时间，减少了低优先级上下文切换次数。综上，GMRNR 相比其他方法减少了开销。

表 1 CPU 开销

Table 1 CPU overhead

	wa/%	id/%	us/%	sy/%	in	cs
LEDF	7	38	50	12	11314	138628
ISACS	5	38	49	13	9802	104722
BOOST	5	40	47	13	10742	129834
MDPSS	3	37	49	14	9013	99743
GMRNR	1	33	58	9	7423	91482

(3)网络访问性能测试

从图 7 可以看出，随着虚拟机数量的增加，在不同大小请求的情况下，LEDF 由于本身并不对优先级进行区分，而是按照先到先得的控制机制执行请求，因此随着资源逐渐饱和，其响应时间无法达到要求的目标，并且因为网络访问请求的连续性，资源饱和状态下的请求响应时间会有所增加。相比 LEDF，ISACS 的控制机制在资源饱和后通过增加虚拟 CPU 分担请求处理，所以部分请求的响应时间有所降低。但是不区分优先级的缺点使得以上两种方法都无法正常保证高优先级的响应时间要求。GMRNR 由于聚合等待作用，相比以上两种方法都凸显了优先级分类的优势，保证了高优先级请求的响应时间。随着资源逐渐饱和，GMRNR 由于自身的控制机制对请求进行聚合响应，当请求数据大小不同，读写文件请求数据较大而聚合队列的聚合长度较小时请求无法聚合，因此，在保证 S 级请求的响应时间的基础上，A 级请求的响应时间基本上可以得到保证，但是这使得 B 级请求占用资源减少，响应时间有所增加。

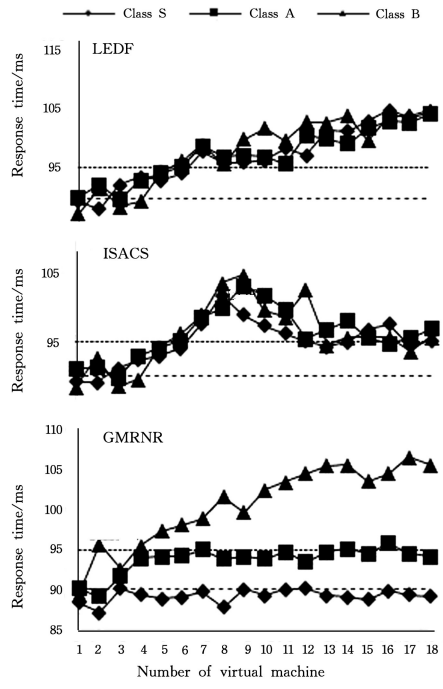

图 7 LEDF，ISACS 和 GMRNR 方法访问磁盘文件的响应时间

Fig. 7 Response time of disk file access for LEDF, ISACS and GMRNR

从图 8 可以看出，BOOST 在不考虑最低优先级请求的情况下，基本上可以保证 A 级请求的请求响应时间。而 MDPSS 由于优先级机制的不同，使得低优先级请求因等待时间过长而提升为高优先级请求，优先执行，缩短了请求的响应时间。

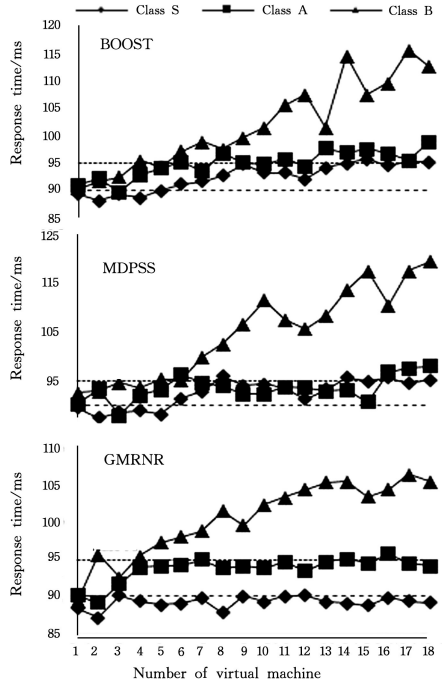

图 8 BOOST，MDPSS 和 GMRNR 方法访问磁盘文件的响应时间

Fig. 8 Response time of disk file access for BOOST, MDPSS and GMRNR

以上两种方法由于缺乏对请求传输的合理控制,在资源饱和时,有一部分 A 级请求很难达到要求的请求响应时间。GMRNR 则凸显了控制机制的作用,分类聚合最大限度地保证了资源饱和状态下的请求响应,保证了高优先级请求的执行。但是在访问磁盘时由于请求数据过大且密集,可能无法聚合而使得部分低优先级请求的响应时间增加。

通过以上对比可知,GMRNR 在区分优先级的基础上增加了聚合控制机制,在资源内部控制请求,在请求大小不同的情况下仍然可以保证高优先级请求的响应时间。

结束语 本文针对资源的访问控制,发现很少有研究通过对网络 I/O 请求的直接控制来保证响应时间,且虚拟化技术大多采用半虚拟化框架 Virtio。通过分析 TAM 方法在 Virtio 中聚合的作用,我们产生在 Virtio 建立门控机制这一思路,提出了实时网络 I/O 请求门控机制(GMRNR)。GMRNR 借助 TAM 方法的聚合作用在 Virtio 前端 virtio-net 模块中设立,将请求按照其响应时间指标分级,采用计时器和聚合队列长度控制不同级别请求通过 Virtio 的时刻和聚合频率,保证各优先级级别请求的响应时间。实验测试表明,GMRNR 能够区分网络 I/O 请求优先级,不论资源是否充足,均可以保证较高优先级请求的响应时间。

GMRNR 也存在一定的局限性:首先,当 3 类优先级请求数量不均衡、高优先级请求数量过大时,可能由于其资源的占用率过高而使得较低优先级请求无法全部达到响应时间要求;其次,GMRNR 内部的聚合门控机制中聚合程度无法自适应,不能根据请求大小的不同自适应地改变聚合队列长度和计时器时长,从而自适应地控制聚合。未来的研究可以从机制内部自适应入手进行更为深入的研究。同时,聚合门控机制的思路也可以应用于其他方面的研究中。

参 考 文 献

[1] LIN W W, QI D Y. Survey of Resource Scheduling in Cloud Computing[J]. Computer Science, 2012, 39(10): 1-6.

[2] LI Q, ZHENG X. Research Survey of Cloud Computing [J]. Computer Science, 2011, 38(4): 32-37.

[3] CHEN Y P, LIU B, LIN W W, et al . Survey of Cloud-edge Collaboration[J]. Computer Science, 2021, 48(3): 259-268.

[4] ZHANG J. Study on Cloud Computing SLA[J]. Telecommunications Network Technology, 2012(2): 7-10.

[5] GUL B, KHAN I A, MUSTAFA S, et al. CPU and RAM Energy-based SLA-aware Workload Consolidation Techniques for Clouds[J]. IEEE Access, 2020, 8: 62990-63003.

[6] NASTIC S, MORICHETTA A, PUSZTAI T, et al. SLOC: Service Level Objectives for Next Generation Cloud Computing[J]. IEEE Internet Computing, 2020, 24(3): 39-50.

[7] LI Q, LI Y, XU B B. QoS-Guaranteed Dynamic Resource Provision in Internet Data Centers[J]. Chinese Journal of Computers, 2014, 37(12): 2395-2407.

[8] SOPIN E S, GORBUNOVA A V, GAIDAMAKA Y V, et al. Analysis of Cumulative Distribution Function of the Response Time in Cloud Computing Systems with Dynamic Scaling[J]. Automatic Control and Computer Sciences, 2018, 52(1): 60-66.

[9] ROSENKRANTZ S, LI H, ENGANTI P, et al. JADE: Tail-Latency-SLO-Aware Job Scheduling for Sensing-as-a-Service[C]// 2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC). IEEE, 2020: 366-373.

[10] MOSA A, SAKELLARIOU R. Dynamic Virtual Machine Placement Considering CPU and Memory Resource Requirements [C]// 2019 IEEE 12th International Conference on Cloud Computing(CLOUD). IEEE, 2019: 196-198.

[11] SUN X, LI Q Z, ZHAO P, et al. An Optimized Replica Distribution Method for Peer-to-Peer Network[J]. Chinese Journal of Computers, 2014, 37(6): 1424-1434.

[12] HOU S, XU S C. A Resource Utilization Threshold based Virtual Machine Allocation Strategy[J]. Journal of Chongqing University of Posts and Telecommunications(Natural Science Edition), 2019, 31(6): 123-130.

[13] WANG Y D, YANG J H, XU C, et al. Survey on Access Control Technologies for Cloud Computing[J]. Journal of Software, 2015(5): 1129-1150.

[14] NAMASUDRA S. Data access control in the cloud computing environment for bioinformatics[J]. International Journal of Applied Research in Bioinformatics (IJARB), 2021, 11(1): 40-50.

[15] POPA L S, YU M, KO S, et al. CloudPolice: Taking Access Control out of the Network[C]// Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010: 1-6.

[16] ZHANG W T. Based on I/O Performance of Virtual Machine Resource Scheduling Algorithm Research[D]. Wuhan: Huazhong University of Science and Technology, 2013.

[17] ZHOU J F. A Study of Virtual Machine Network Bandwidth Dynamic Regulation Mechanism[D]. Wuhan: Huazhong University of Science and Technology, 2012.

[18] MOUZAKITIS A, PINTO C, NIKOLAEV N, et al. Lightweight and Generic RDMA Engine Para-Virtualization for the KVM Hypervisor[C]// International Conference on High Performance Computingand Simulation. IEEE, 2017: 737-744.

[19] KUKREIA G, SINGH S. Virtio based Transcendent Memory [C]// IEEE International Conference on Computer Science and Information Technology. IEEE, 2010.

[20] ARA G, LAI L, CUCINOTTA T, et al. A Framework for Comparative Evaluation of High-Performance Virtualized Networking Mechanisms[J]. Cloud Computing and Services Science, 2021, 1399: 59.

[21] LIU Y Y, NIU B N. Optimizing Network Performance Based on Para-virtualization Virtio Framework[J]. Journal of Chinese Computer Systems, 2018, 39(1): 105-110.

[22] CHENG S X. Research and Optimization of I/O Performance Bottleneck in Embedded Virtualization Environment[D]. Shanghai: Shanghai Jiao Tong University, 2015.

[23] SU X, LI Y F, ZONG N, et al. Network Real-time Scheduling

Algorithm based on Multi-feature Dynamic Priority[J]. Journal on Communications, 2020, 41(5): 159-167.

[24] QIU H, BANERJEE S S, JHA S, et al. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices[C]// 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20). 2020: 805-825.

[25] LEE J, YU H. I/O Strength-Aware Credit Scheduler for Virtualized Environments[J]. Electronics, 2020, 9(12): 2107.

[26] ZHANG C, YU M, YAN F. Enabling Cost-Effective, SLO-Aware Machine Learning Inference Serving on Public Cloud[J/OL]. IEEE Transactions on Cloud Computing. https://ieeexploreieee.53yu.com/abstract/document/9132666.

[27] ZHOU M S, DONG X S, CHEN H, et al. Dynamically Fine-grained Scheduling Method in Cloud Environment[J]. Journal of Software, 2020, 31(12): 3981-3999.

[28] LIU H. Research and Implementation of Storage Architecture based on Load Balancing[D]. Jinan: Shandong University, 2011.

[29] RAJAVEL R, MALA T. Achieving Service Level Agreement in Cloud Environment using Job Prioritization in Hierarchical Scheduling[C]// Proceedings of the International Conference on Information Systems Design and Intelligent Applications (INDIA 2012). Springer, 2012: 547-554.

[30] CHAPALA Y, REDDY B E. An Enhancement in Restructured Scatter-Gather for Live Migration of Virtual Machine[C]// 2021 6th International Conference on Inventive Computation Technologies (ICICT). IEEE, 2021: 90-96.

[31] KIM J H, JIN H W. Virtio Front-end Network Driver for RTEMS Operating System[J]. IEEE Embedded Systems Letters, 2019, 12(3): 91-94.

[31] RUSSELL R. Virtio: Towards a De-facto Standard for Virtual I/O Devices[J]. ACM SIGOPS Operating Systems Review, 2008, 42: 95-103.

SHEN Hao-xi, born in 1994, postgraduate. His main research interests include cloud computing virtualization technology and service level objective (SLO).

NIU Bao-ning, born in 1964, Ph.D, professor, Ph. D supervisor, is a senior member of China Computer Federation. His main research interests include performance management of DBMS, blockchain, big data management and analysis, etc.

(责任编辑:李亚辉)